PATENT APPLICATION

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Applicants: Degenaro, et al.

Examiner: Choi, Woo H.

Serial No.: 10/776,909

Group: Art Unit 2189

Filed: February 11, 2004

Docket: YOR919990064US2 (8728-258CON)

For: SYSTEM AND METHOD FOR MANAGING CACHABLE ENTITIES

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313

APPEAL BRIEF

Appeal from Group 2189
F. Chau & Associates, LLC
130 Woodbury Road
Woodbury, New York 11797
TEL: (516) 692-8888
FAX: (516) 692-8889
Attorneys for Appellants

TABLE OF CONTENTS

This Appeal was initially commenced by a Notice of Appeal and Pre-Appeal Brief Request for Review filed on March 6, 2006 in response to a Final Office Action mailed on November 18, 2005 finally rejecting claims 1-51 of the above-identified application. An Appeal Brief was filed on July 24, 2006 in furtherance of the Appeal. In response, prosecution was reopened by issuance of a new Final Office Action mailed on October 18, 2006 (hereinafter, referred to as the "Final Action") for purposes of including previously asserted rejections that were supposedly inadvertently omitted during prosecution leading up to the initial Final Rejections which prompted this Appeal. Applicants reinstated the Appeal in this action by virtue of a Notice of Appeal filed on January 18, 2007, and hereby submit this Appeal Brief in furtherance of the reinstated Appeal.

I. REAL PARTY IN INTEREST

The real party in interest for the above-identified application is International Business Machines Corporation, the assignee of the entire right, title and interest in and to the subject application by virtue of an assignment of recorded in the U.S. Patent and Trademark Office.

II. RELATED APPEALS AND INTERFERENCES

There are no Appeals or Interferences known to Applicant, Applicant's representatives or the Assignee, which would directly affect or be indirectly affected by or have a bearing on the Board's decision in the pending Appeal.

III. STATUS OF CLAIMS

Claims 1-51 are pending, stand rejected and are under appeal. The claims are set forth in the attached Appendix. Claims 1, 18 and 35 are independent claims. Claims 2-17, 41-44 and 49 depend directly or indirectly from base claim 1. Claims 19-34, 45-48 and 50 depend directly or

indirectly from base claim 18. Claims 36-40 and 51 depend directly or indirectly from base claim 35.

IV. STATUS OF AMENDMENTS

No claim amendments have been filed or entered subsequent to the Final Action.

V. SUMMARY OF CLAIMED SUBJECT MATTER

In general, the claimed inventions are directed to systems and methods for managing cacheable entities (e.g., objects that are stored and/or could be stored in a cache) in a data processing system. The claimed inventions for managing cacheable entities are generally based on automated methods for making cache decisions by analyzing program code to determine the desirability of performing cache transactions (e.g., caching or deleting an object or updating a cached object, etc.). For purposes of illustration, the claimed inventions will be described with reference to the exemplary Figures and corresponding text of Appellants' Specification (Spec.), but nothing herein should be unduly construed to limit the scope of the claimed inventions.

Claim 1 recites:

> ***An automated method for managing a plurality of cachable entities, comprising the steps of:***
>
> ***analyzing program code to determine if there is at least one statement which can affect a desirability of performing at least one cache transaction, if the at least one statement is executed;***
>
> ***determining a probability that the at least one statement will execute;***
>
> ***determining the desirability of performing the at least one cache transaction based on the probability that the at least one statement will execute.***

Exemplary embodiments of the invention of claim 1 will be discussed with reference to FIG. 6 and corresponding text, which illustrate an exemplary ***automated method for managing a plurality of cachable entities*** (see generally, Spec. , p. 35, line 11 – p. 39, line 6, for example)

The term "cacheable entities" refers to entities (e.g., objects) that are stored in a cache or that may be stored in a cache (see, Spec., p. 2, line 22 ~ p. 3, line 2).

The method includes ***analyzing program code to determine if there is at least one statement which can affect a desirability of performing at least one cache transaction, if the at least one statement is executed***. In general, this step involves performing a program analysis to detect one or more statements, if any, which can affect the desirability of performing a cache transaction if the statement is executed. Such statements include, for example, statements that can result in object state changes, creation of objects, deletion of objects, etc. For instance, a program can be analyzed to identify or otherwise detect one or more statements (if they exist) which may modify a value of one or more cachable entities (e.g., an object, image file, webpage, etc.) during run-time (see, e.g., Spec. p. 36, lines 5-9; Fig. 6, block 600). In another embodiment, program analysis could be implemented to detect one or more statements which materialize or fetch a value of one or more cacheable entities **(see, e.g., Spec., p. 38, lines 12-15).**

For a detected statement (if any), the process includes ***determining a probability that the at least one statement will execute***. In other words, when a given statement ***which can affect a desirability of performing a cache transaction*** is identified as a result of the program analysis, a determination is made as to the probability (or likelihood) that the given statement will be executed during runtime (See, e.g., Spec., p. 36, lines 9-12, FIG. 6, block 601). For example, if the given statement can modify a value of a cacheable entity, there is a likelihood that one or more cacheable entities will change due to execution of the given statement. The probability of the statement being executed can be determined in various ways. For example, if a detected statement will execute outside of a conditional branch in a program, the probability that the

statement will execute may be "1". If, on the other hand, a detected statement executes within a conditional branch (e.g., if (y > 0) then x = a * b) the probability that the statement will execute can often be determined from program analysis. In the previous example, the compiler might have determined through analysis that "y" is extremely likely to be positive. If so, it would conclude that x has a high probability of changing (see, e.g., Spec. p. 36, lines 9–23).

The claimed process further includes ***determining the desirability of performing the at least one cache transaction based on the probability that the at least one statement will execute***. In other words, the desirability of performing the cache transaction is determined, at least in part, based on the *probability that the given statement* (which was determined during program analysis to be one in which can affect the desirability of performing the cache transaction.) *will execute.* **(See, e.g., FIG. 6, blocks 602, 603, 604, and Spec. p. 37, lines 1-15).**

For example, in an exemplary embodiment where the detected statement is one that may modify a value of one or more cachable entities, the probability that the statement will execute may correspond to the likelihood that one or more cachable entities will change due to execution of the statement (see, p. 36, lines 9-14). The desirability of performing a cache transaction is based, for exemplar, on whether the likelihood of change meets a predefined threshold (step 602). If it is determined that the likelihood of change exceeds the threshold (affirmative determination in step 602), the system (i) may not be in favor of caching one or more uncached entities and/or the system (ii) may be in favor of invalidating or updating one or more cached entities (step 603). On the other hand, if it is determined that the likelihood of change does not exceed the threshold (negative determination in step 602), the system (i) may

be in favor of caching one or more uncached entities and/or the system (ii) may not be in favor of invalidating or updating one or more cached entities (step 604). (See, e.g., p. 37, lines 1-15).

Claim 42 recites:

The method of claim 1, further comprising determining said probability based on a likelihood of a value of a cachable entity changing.

As noted above, claim 1 recites ***determining a probability that the at least one statement will execute.*** Claim 42 further refines the "determining" step in that the *probability determination is based on a likelihood of a value of a cachable entity changing*. For example, if a detected statement executes within a conditional branch (e.g., if (y > 0) then x = a * b) the probability that the statement x = a * b will execute can be determined based on the probability of change of the value x. For example, where a compiler might have determined through program analysis that the value of "y" is extremely likely to be positive, it would conclude that x has a high probability of changing **(see, e.g., Spec. p. 36, lines 9~23; FIG. 6, step 601).** In other words, in this example, the likelihood of change of "x" due to positive "y" (as determined through program analysis) corresponds to the likelihood of execution of the statement x = a * b since the statement will be executed to compute a new value of x).

Claim 43 recites:

The method of claim 42, further comprising caching a value of said cachable entity, if said probability is less than or equal to a threshold. (See, e.g., Spec. p. 37, lines 10-15)

Claim 44 recites:

The method of claim 42, further comprising invalidating or updating a value of said cachable entity, if said probability is greater than or equal to a threshold. (see, e.g., Spec. p. 37, lines 5-10).

Claims 18, 46, 47 and 48 are similar in subject matter to the claimed inventions 1, 42, 43, and 44, respectively, as discussed above, and the same summary explanation applies.

Claim 18 recites:

A program storage device readable by a machine, tangibly embodying a program of instructions executable by the machine to perform method steps for managing a plurality of cachable entities, the method steps comprising: (see generally, Spec. , p. 6, line 1 ~ p. 7, line 18; p. 35, line 11 ~ p. 39, line 6, for example)

analyzing program code to determine if there is at least one statement which can affect a desirability of performing at least one cache transaction, if the at least one statement is executed; (see, e.g., Spec. p. 36, lines 5-9; Fig. 6, block 600).

determining a probability that the at least one statement will execute; (See, e.g., Spec., p. 36, lines 9-12, FIG. 6, block 601).

determining the desirability of performing the at least one cache transaction based on the probability that the at least one statement will execute. (See, e.g., p. 37, lines 1-15).

Claim 46 recites:

The program storage device of claim 18, further comprising instructions for determining said probability based on a likelihood of a value of a cachable entity changing. (see, e.g., Spec. p. 36, lines 9~23; FIG. 6, step 601).

Claim 47 recites:

The program storage device of claim 46, further comprising instructions for caching a value of said cachable entity, if said probability is less than or equal to a threshold. (See, e.g., Spec. p. 37, lines 10-15)

Claim 48 recites:

The program storage device of claim 46, further comprising instructions for invalidating or updating a value of said cachable entity, if said probability is greater than or equal to a threshold. (See, e.g., Spec. p. 37, lines 5-10)

Claim 35 recites:

A system for managing a plurality of cachable entities, comprising:

a program analyzer to analyze program code and determine if there is at least one statement which can affect a desirability of performing at least one cache transaction, if the at least one statement is executed; (See, e.g., Spec. , p. 35, line 11 ~ p. 39, line 6; p. 9, lines 6-16, FIG. 1, blocks 102, 103).

the program analyzer determining a probability that the at least one statement will execute and determining the desirability of performing the at least one cache transaction based on the probability that the at least one statement will execute; and (see, e.g., Spec. p. 36, lines 5-9; Fig. 6, block 600; p. 36, lines 9-12, block 601).

a cache manager for performing the at least one cache transaction if it is determined to be desirable. (See, e.g., p. 37, lines 1-15; and p.10, line 11 ~ p. 12, line 18; FIG. 1, block 106).

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

Claim Rejections - 35 U.S.C. § 101

A. Claims 1-3, 5, 8-11, 14, 18-20, 22, 25-28, 42 and 46 are rejected as being directed to non-statutory subject matter.

B. Claims 1-48 are rejected based on nonstatutory double patenting over claims 1-37 of U.S. Patent No. 6,725,333.

Claim Rejections - 35 U.S.C. § 112

C. Claims 42-44 and 46-48 stand rejected under 35 § U.S.C., first paragraph, for supposedly failing to comply with the written description requirement.

Claim Rejections - 35 U.S.C. § 102

D. Claims 1, 5, 10, 18, 22, 27, 35, 39, 40, 41 and 45 stand rejected under 35 U.S.C. § 102(b) as being unpatentable over U.S. Patent No. 5,940,857 to Nakanishi.

E. Claims 1, 2, 4-6, 8, 18, 19, 21, 23, 25, 35, 36, 38-41 and 45 stand rejected under 35 U.S.C. 102(b) as being unpatentable over U.S. Patent No. 5,774,685 to Dubey.

F. Claims 1-8, 10, 14-25, 27 and 31-48 stand rejected under 35 U.S.C. § 102(b) as being unpatentable over Cytron.

Claim Rejections - 35 U.S.C. § 103

G. Claims 9 and 26 stand rejected as being unpatentable over Cytron in view of U.S. Patent No. 6,073,129 to Levine, et al.

VII. ARGUMENTS

Claim Rejections - 35 U.S.C. § 101

A. The Claims are Directed to Statutory Subject Matter

Appellants respectfully submit that claims 1-3, 5, 8-11, 14, 18-20, 22, 25-28, 42 and 46 are clearly directed to statutory subject matter. At the very least, the Final Action utterly fails to present a *prima facie* showing that such claims 1, 18, or 35 are "abstract ideas" directed to non-statutory subject matter. As can be readily gleaned from the "prima facie" analysis set forth on page 2 of the Final Action, the 101 utility rejections are based essentially on the Examiner's finding that:

> ***"The language of the claim raises a question as to whether the claims are directed merely to an abstract idea that is not tied to a technological art … which would result in a practical application producing a concrete, useful and tangible result …"*** **and that** ***"the claims seem to be directed to a method of planning/preparing with no practical application of the plan"*****, which "*****amounts to a machined manipulated abstract idea.*****"**

Appellants respectfully contend that the above findings are erroneous as a matter of law and fact.

As noted in section 2106 of the MPEP:

> A process that consists solely of the manipulation of an abstract idea is not concrete or tangible. See *In re Warmerdam*, 33 F.3d 1354, 1360, 31 USPQ2d 1754, 1759 (Fed. Cir. 1994). See also *Schrader*, 22 F.3d at 295, 30 USPQ2d at 1459. Office personnel have the burden to establish a *prima facie* case that the claimed invention as a whole is directed to solely an abstract idea or to manipulation of abstract ideas or does not produce a useful result. Only when the claim is devoid of any limitation to a practical application in the technological arts

> should it be rejected under 35 U.S.C. 101. . . Further, when such a rejection is made, Office personnel must expressly state how the language of the claims has been interpreted to support the rejection. (See M.P.E.P. 2106 (II)(A), 8th Edition, August 2001, Rev. 2005)

In the case at bar, the Examiner's "prima facie" case is premised on overly broad, arbitrary claim interpretations that fail to consider, and ignore the clear express language of the claimed inventions. Indeed, in the absence of any supporting explanation, the Examiner's interpretation of the claims being directed to an "abstract idea" of "planning/preparing" with no practical application of the plan, is erroneous on its face. The Examiner's *prima facie* analysis is seemingly premised on an initial step of characterizing the claimed inventions in a highly generic, abstract category of "planning/preparation" and then using this "abstract" interpretation as the basis for the claim rejection, without due consideration given to the actual claim language. Clearly, this analysis is improper as a matter of law- if this was the proper legal standard, then every claimed invention could be construed according to some abstract category and rejected as being directed to non-statutory subject matter based on its abstract concept.

In contrast to the Examiner's findings, Appellants respectfully assert that when properly construed in view of the specification, the claimed inventions are not merely abstract ideas not tied to a technological art, but rather directed, in general, to useful and practical automated methods for managing cacheable entities using program analysis for aiding in making cache decisions. Moreover, the claimed inventions are clearly tied to computer-based technological arts of data processing and cache management, for example. In fact, the Examiner apparently recognized the scope and practical application of the claimed inventions, as evidenced by the Examiner's prior art rejections using cited art in the technical fields of computer caching systems and methods.

Another fundamental flaw in the Examiner's 101 arguments is that the Examiner fails to recognize that claim 18 is not directed to a "method of planning …", but rather *a program storage device readable by a machine, tangibly embodying a program of instructions executable by the machine* to perform the method steps recited in claim 18. In other words, the Examiner's analysis and findings are not on point as the claimed inventions are directed NOT to a process, *per se*, but rather a device- a program storage device having instructions for performing the claimed process steps.

As explained in MPEP 2106(IV)(B)(1), descriptive material can be characterized as either "*functional descriptive material*" or "nonfunctional descriptive material." In this context, "functional descriptive material" consists of data structures and computer programs which impart functionality when employed as a computer component. When functional descriptive material is recorded on some computer-readable medium it becomes structurally and functionally interrelated to the medium and will be statutory in most cases since use of technology permits the function of the descriptive material to be realized. Here, at the very least, claim 18 should be properly viewed as being directed to program storage devices having functional description material recorded thereon, and is thus, directed to statutory subject matter. The Examiner's rejections fail to acknowledge or even address this point. For at least the above reasons, the claim rejections under 101 should be reversed.

B. The Double Patenting Rejections Will Not Be Challenged on This Appeal

For purpose of this Appeal, Applicant will concede to the propriety of the double patenting rejection and will file a Terminal Disclaimer in accordance with 37 C.F.R. 1.321 to overcome the double patenting rejection, if the Board reverses the claim rejections upon

disposition of this Appeal, or otherwise upon final disposition of prosecution of this application, if necessary.

C. **The Claimed Inventions Comply with the Written Description Requirement.**

Appellants respectfully submit that claims 42-44 and 46-48 comply with the written description requirement

The written description requirement is met when an applicant shows possession of the claimed invention by describing the claimed invention with all of its limitations using "such descriptive means as words, structures, figures, diagrams, etc., that fully set forth the claimed invention." *Lockwood v. American Airlines, Inc.* 107 F.3d 1565, 1572 (Fed. Cir. 1997). It is well established that the subject matter of the claim need not be described literally (i.e., using the same terms or *in haec verba*) in order for the disclosure to satisfy the description requirement. See *Purdue Pharma L.P. v. Faulding Inc.*, 230 F.3d 1320, 1323 (Fed. Cir. 2000). "If a person of ordinary skill in the art would have understood the inventor to have been in possession of the claimed invention at the time of filing, even if every nuance of the claims is not explicitly described in the specification, then the adequate description is met." *In re Alton*, 76 F.3d at 1175 (see also *Vas-Cath*, 935 F.2d at 1563).

Claims 42 and 46

Claims 1 and 18 recite "*determining a probability that the at least one statement will execute*". Claims 42 and 46 (which depend from claims 1 and 18, respectively) recite "further comprising *determining said* probability based on a likelihood of a value of a cachable entity changing". In other words, claims 42 and 46 can be interpreted to mean *that the probability that*

a statement will execute is determined based on a likelihood of a value of a cachable entity changing.

On page 4 of the Final Action, the Examiner essentially contends that claims 42 and 46 do not meet the written description requirement because Page 36, lines 14-23 of the Appellants' Specification discloses that "the probability of a value of a cacheable entity changing is based on the probability that a statement will execute, not the other way around as claimed." The Examiner's contention is incorrect insofar as the Examiner asserts that there is no support for the "other way around as claimed". Indeed, the cited section provides clear support for claims 42 and 46. For example, page 36, lines 17-23 of Spec. states:

> For example, if a statement is executed outside of a conditional branch in a program, the probability that the statement will execute is often 1. If, on the other hand, a statement executes within a conditional branch (e.g., if (y > 0) then x = a * b) the probability that the statement will execute can often be determined from program analysis. In the previous example, the compiler might have determined through analysis that "y" is extremely likely to be positive. If so, it would conclude that x has a high probability of changing.

In other words, the above example fairly supports *determining a* probability that a statement (x=a*b) will execute based on a likelihood of a value "x" of a cachable entity (x) changing given the likelihood of "y" being positive (as determined through program analysis), resulting in likelihood of execution of the conditional statement x=a*b. Moreover, in the above example, the likelihood of the conditional statement (x=a*b) being executed is based on the probability (as determined through program analysis) of the value of "y" being positive, thus resulting in execution of the conditional statement.

Claims 43, 44, 47, 48:

The Examiner further contends (on page 4 of the Final Action) essentially that claims 43,

44, 47 and 48 do not meet the written description requirement because:

> caching, updating and invalidating are to be performed depending on the likelihood of a value of a cachable entity changing exceeding or not exceeding a threshold, as opposed to the probability of a statement execution exceeding or not exceeding a threshold (specification page 37, lines 1~15).

Claims 42 and 46 essentially recites "*determining said* probability based on a likelihood of a value of a cachable entity changing". Claims 43 and 46 (which depend from claim 42 and 46, respectively) essentially recite "***caching a value of said cachable entity, if said probability is less than or equal to a threshold.*** Claims 44 and 47 (which depend from claims 42 and 46, respectively), essentially recite ***invalidating or updating a value of said cachable entity, if said probability is greater than or equal to a threshold.***

The basis for this rejection is not clear and seemingly meritless. On one level, the Examiner fails to recognize that claims 43/44 depend from claim 42 and that claims 47/48 depend from claim 46, wherein claims 42 and 46 essentially recite determining *said probability based on a likelihood of a value of a cachable entity changing*. Thus, the term "said probability" in claims 43/44, 47/48 is limited by the language of claims 42/46 where said probability is *based on a likelihood of a value of a cachable entity changing*. In such case, there is clear support caching/updating/invalidating steps are performed depending of said probability (*likelihood of a value of a cachable entity changing* likelihood), which is explicitly described in Applicants' specification.

On another level, the Examiner fails to recognize the teachings on page 36, lines 1-15 of Spec., where a probability is determined which represents the likelihood that the detected statements will be executed. In the context of the claimed invention where the detected

statements are those that can possibly modify a value of a cachable entity if the statement is executed, the likelihood that the value of a cachable entity can change is based, at least in part, on the likelihood that the statement (modifying the value) will execute. In view of the above, it is clear that Examiner's written description rejections are legally deficient as a matter of law and constitute reversible error.

Claim Rejections - 35 U.S.C. § 102

For a claim to be anticipated under 35 U.S.C. § 102, all elements of the claim must be found in a single prior art reference and there must be no difference between the claimed invention and the reference disclosure, as viewed by a person of ordinary skill in the field of the invention (see, e.g., Scripps Clinic & Research Found. v. Genentech Inc., 927 F.2d 1565, 1576, 18 USPQ2d 1001, 1010 (Fed. Cir. 1991). The single prior art reference must disclose all of the elements of the claimed invention functioning essentially in the same manner (see, e.g., Shanklin Corp. v. Springfield Photo Mount Corp, 521 F.2d 609 (1st Cir. 1975)).

D. The Teachings of *Nakanishi* Do Not Support the Anticipation Rejections

In the case at bar, Nakanishi is legally deficient to establish a prima facie case of anticipation against any of claims 1, 5, 10, 18, 22, 27, 35, 39, 40, 41 and 45. The basis for the anticipation rejections is set forth on pages 4-5 of the Final Action, where the Examiner offers a general, omnibus rejection for claims 1, 5, 10, 18, 22, 27, 35, 39, 40, 41 and 45, which for the most part, fails to explain how Nakanishi teaches many of the claimed features.

At the very least, Nakanishi does not disclose or remotely suggest the inventions of claim 1, 18 or 35. As will be explained hereafter, the Examiner's rejections are based on a strained

interpretation of Nakanishi in an attempt to fit Nakanishi's teaching to the claim elements. When properly construed, the teachings of Nakanishi fail to anticipate the claimed inventions.

On a fundamental level, Nakanishi does not disclose or suggest various elements of claims 1 and 18. For instance, Nakanishi does not disclose or suggest ***analyzing program code to determine if there is at least one statement which can affect a desirability of performing at least one cache transaction, if the at least one statement is executed***, as essentially recited in claims 1 and 18. The Examiner relies on the Abstract and Col. 4, lines 22-44 of Nakanishi as disclosing this feature. However, the Examiner's reliance is misplaced.

Nakanishi discloses (Abstract and Col. 4, lines 22-44) an instruction cache memory that includes an advance read function, wherein an instruction analysis process for predicting whether it is necessary to read out a next block of instructions from a main memory by analyzing the instructions contained in a block of instructions that was currently read from main memory and presently being transferred to an instruction cache (Abstract). In particular, with instruction analysis function disclosed by Nakanishi, the instructions included in a read block being read out from the main memory are analyzed to predict if it is necessary or not to read out a next block of instruction (which is a block succeeding the read block) from main memory (see, Col. 4, lines 22-30). The purpose of this advance read function is to detect whether the read block of instructions corresponds to a branch predict instruction which is predicted to result in a branch operation that branches to a region outside of the read block and the next block (see, e.g., Col. 29, lines 14-22), thereby reducing the need to read out a next block unnecessarily when there is a likelihood of a branch operation (see, e.g., Col. 4, lines 35-59).

The Examiner contends that Nakanishi teaches "an instruction is analyzed to determine whether it is desirable to cache the instructions in the next block" and that such teaching reads on the claimed step of ***analyzing program code to determine if there is at least one statement which can affect a desirability of performing at least one cache transaction, if the at least one statement is executed*** . This finding is fundamentally flawed for various reasons.

First of all, Nakanishi clearly teaches that the program analysis is performed on a currently read block of instructions to determine the possibility of a branch operation and determine whether or not a next succeeding block of instructions should be read from main memory, i.e., to implement an advance read function. In other words, the purpose of the program analysis of Nakanishi is not to determine, per se, if there is an instruction that *can affect a desirability of performing a cache transaction, if the statement is executed*, as claimed in claims 1 and 18, but rather whether there in an instruction in a currently read block that enables prediction of whether or not a next succeeding block of instructions should be fetched from main memory. In this regard, the fact that Nakanishi discloses that the fetched blocks of instructions are stored in an instruction cache to service CPU requests is essentially irrelevant as the focus of program analysis in Nakanishi is to determine whether or not to read a next block from memory in view of possible branch operations. The program analysis of Nakanishi is fundamentally different in purpose and function from the claimed program analysis functionality.

Another flaw in the Examiner's analysis is that the Examiner parses the claim language in a way that ignores some claim language and renders the analysis illogical in view of the claim language and applied teachings of Nakanishi. For instance, in the above analysis, the Examiner fails to address the specific claim language that the program code is analyzed to determine if

there is at least one statement which can affect a desirability of performing at least one cache transaction, if the at least one statement is executed. If the detected statement is not executed, it does not affect a desirability of performing a transaction. The Examiner does not explain how Nakanishi teaches this aspect of the claimed invention. In Nakanishi, the read out blocks are presumed to be executed by the CPU, so there is no notion of determining if a statement in a read block being analyzed with execute.

In this regard, there is no merit to the Examiner's contention that Nakanishi teaches (in Col. 27, lines 7-14), the claimed step of ***determining a probability that the at least one statement will execute***, as claimed in claims 1 and 18. As noted above, Nakanishi teaches that program analysis is performed on a block by block basis and is performed on a currently read block of instructions. Nakanishi teaches (FIG. 12) that the instruction analysis section (5) comprises a plurality of branch instruction analysis circuits for detecting whether a branch instruction is included or not, determining the type of branch instruction, if included, and detecting the branch destination address or offset, etc. (see, Col. 18, lines 56-65). The instruction analysis section (5) analyzes the content of a read block of instruction to determine if there is a branch instruction or not, and if there is a branch instruction, and branch destination address is calculated (see, e.g., Col. 19, lines 12-20).

In view of the above, it is glaringly clear that Nakanishi does not disclose that the read block is analyzed to detect statements that if executed, would affect a desirability of performing a cache transaction and then determine a probability that the (detected) statement will execute.

Moreover, Examiner's reliance on Col. 27, lines 7-14 is wholly misplaced. Nakanishi teaches that a branch instruction may have a branch information bit that specifies whether to read

out a branch destination block from main memory. The instruction analysis section (5) merely detects if the branch information bit of a branch instruction (in the block being analyzed) is either set or not set, to thereby output a branch predict signal on the basis thereof. Again, there is no notion in the cited section of determining a probability that a branch instruction (detected in the analyzed block) will be executed or not. Nakanishi teaches that the detected branch instruction statement will execute, but predicts the results of the branching so as to be able to prefetch the branch destination block from main memory (see, Col. 6, lines 42-50).

According, for at least the above reasons, it appears that Examiner's reliance on Nakanishi's "instruction analyzing" step is improper and out of context, as such analysis process is dissimilar in function and purpose as the claimed invention. As such, the anticipation rejections based on Nakanishi should be reversed.

E. The Teachings of *Dubey* Do Not Support the Anticipation Rejections

In the case at bar, Dubey is legally deficient to establish a *prima facie* case of anticipation against any of claims 1, 2, 4-6, 8, 18, 19, 21, 23, 25, 35, 36, 38-41 and 45. The anticipation rejections based on Dubey are set forth on pages 6-7 of the Final Action. In formulating the anticipation rejections, the Examiner once again offers an omnibus anticipation analysis that is commonly applied to multiple claims, including independent claims 1, 18 and 35, without differentiation. In this respect, Appellants will address the anticipation rejections of claim 1, 17 and 35 together.

In general, Dubey is related to a method for performing a compile-time analysis of a program, in which an instruction "STOUCH" enables prefetching data and storing the prefetched

data in a cache based on compile-time speculations associated with conditional branches. In short, Dubey discloses a scheme to prefetch data or instructions and place them into a cache to prevent penalties associated with cache misses during program execution.

Examiner relies on Dubey at Col. 3, line 58- Col. 4, line 4 as disclosing ***analyzing program code to determine if there is at least one statement which can affect a desirability of performing at least one cache transaction, if the at least one statement is executed*** . Dubey specifically discloses in the cited section:

> a prefetching scheme [that] is based on compile-time analysis to determine specific locations within a program where instructions or data cache misses are likely to be encountered at run-time. Non-sequential branch targets or irregular data accesses are typical examples of cache accesses which are likely to cause a cache-miss. In order to minimize the performance impact of such cache-misses, the compiler identifies a set of points where it can provide an early hint to the run-time hardware to initiate a prefetch to hide the potential miss penalty associated with the cache accesses identified before. … such static program locations are referred to as "prefetch points.".

In other words, although Dubey discloses analyzing program code, such analysis is for the purpose of identifying "prefetch points" or points at which instruction or data cache misses are likely to occur at run-time, such that data or instructions can be prefetched from main memory (see, Col. 3, lines 27-38) and thus to reduce latency of possible cache-misses.

Even assuming, arguendo, that Duby's program analysis functionality may be found to read on the claimed "analyzing " step, Dubey does not disclose ***determining a probability that the at least one statement*** (detected by the program analysis) ***will execute,*** or ***determining the***

desirability of performing the at least one cache transaction based on the probability that the at least one statement (detected by the program analysis) ***will execute.***

The Examiner contends that Dubey teaches ***determining a probability that the at least one statement*** (detected by the program analysis) ***will execute*** because "speculative prefetches associated with conditional branches by its very nature is a determination that is it probable that a particular branch will be taken". The Examiner further contends that Dubey teaches (in Col. 4, lines 31-44) ***determining the desirability of performing the at least one cache transaction based on the probability that the at least one statement*** (detected by the program analysis) ***will execute*** because the compiler determines the desirability of prefetching data based on a speculation that the prefetched data will actually be needed, i.e., based on the probability or likelihood that the branch that requires the data will be taken, and inserts a STOUCH instruction a prefetch point. For the following reasons, both contentions are meritless.

In the first instance, the Examiner's finding that "*speculative prefetches associated with conditional branches by its very nature is a determination that it is probable that a particular branch will be taken*" is seemingly irrelevant and does not teach or suggest ***determining a probability that the at least one statement will execute.*** As noted above, Dubey teaches a pre-compile process where a compiler analyzes a static sequence of code (see, e.g. FIG. 1) to identify specific locations within the sequence where cache misses are like to occur (e.g., block B10 store instruction). Based on these specific locations within the static sequence, the compiler identifies "prefetch points" at which to initiate prefetch operations associated with cache accesses at the identified (possible cache miss) locations. The compiler then inserts STOUCH instructions at the

prefetch points. (see, Col. 9, lines 36-56). For instance, in FIG. 1, a STOUCH instruction associated with a prefetch operation for block B10 is inserted as block B1.

As illustrated in FIG. 2, the STOUCH instruction includes an address block (13) that specifies an address for prefetching an instruction or data block from main memory into a cache block upon execution of the STOUCH instruction, and a plurality of compile-time conditions (15) associated with conditional branches between the time the prefetch request is initiated and the time the prefetched data is actually needed during run-time. These conditions specify the sequence of branches (control flow) that if dynamically followed during run-time, will lead to the block for which the prefetch was performed (i.e., the block that was identified at compile time as one which is likely to cause a cache-miss at runt time).

For instance, in FIG. 1, block B10 (store instruction) may be identified at compile time as one which is likely to cause a cache miss at run time, and a prefetch point in block B1 is determined for B10. In FIG. 1, there are three conditional branches between block B1 and block B10, i.e., conditional branches at the end of blocks B1, B2 and B3. A speculative prefetch operation can be performed at block B1 for block B10, and the intervening branch outcomes (as specified by the conditions) are evaluated at runtime to determine if the dynamic control flow will or wont flow to block B10. If not, the prefetched data/instruction is discarded from cache, whereas the prefetched data/instruction will be maintained in cache as the conditions continue to evaluate true (see, Col. 4, line 7, through Col. 5, line 66).

In this regard, Dubey does not teach ***determining a probability that the at least one statement*** (detected by the program analysis) ***will execute.*** As noted above, Dubey detects statements where cache misses are likely to occur and then generates a STOUCH instruction

(which is inserted at a prefetch point) that specifies conditions of specific conditional branch outcomes that would result in a run-time control flow leading from the prefetch point to the regular access point. With this process, when a given statement is detected, there is no determination as to the probability that the statement will execute. In contrast, it is assumed that the statement will execute (i.e., speculative prefetch) and the STOUCH instruction for the given statement is generated to initiate a speculative prefetch and specify the compile time conditions that are evaluated against the run-time outcomes to determine whether to discard or maintain the prefetch cache instruction/data. In other words, for a detected statement, Dubey does not determine a likelihood that the statement will execute, rather Dubey specifies that the branch sequence leading from the prefetch point to the block for which the prefetch was initiated, irrespective of whether such sequence is probable or not probable.

Furthermore, in view of the above, it necessarily follows that Dubey does not teach ***determining the desirability of performing the at least one cache transaction based on the probability that the at least one statement*** (detected by the program analysis) ***will execute.*** The Examiner contends that this claim limitation is met based on a characterization of Dubey as teaching:

> "the compiler determines the desirability of prefetching data based on a speculation that the prefetched data will actually be needed, i.e., based on the probability or likelihood that the branch that required the data will be taken, and inserts an STOUCH instruction at a prefetch point." (see, page 6 of Final Action)

However, this contention is seemingly based on an improper reading of Dubey. Indeed, Dubey teaches that compile time analysis is performed to determine locations within a program wherein cache misses are likely to occur at run-time. In other words, Dubey program analysis determines cache accesses that are likely to cause a cache-miss (Col. 3, lines 60-65), and

compiles prefetch instruction (STOUCH) to prefetch data/instruction into a cache. In this regard, there is no merit to the Examiner's characterization of Dubey teaching "a desirability of prefetching data based on a *speculation* that the prefetched data will actually be needed, i.e., based on the *probability or likelihood* that the branch that required the data will be taken . . ". First of all, the term speculation is not synonymous with probability or likelihood. In Dubey, the term "speculation" refers to the set of conditions associated with a STOUCH instruction that specify the control path of a specific branch outcome that is used to determine whether to maintain or discard the prefetched data/instruction based on actual run-time control flow. These speculations or conditions are not "probabilities of statement execution", but merely conditions that are evaluated during run-time.

For at least the above reasons, claims 1, 18 and 35 are clearly not anticipated by Dubey.

F. The Teachings of *Cytron* Do Not Support the Anticipation Rejections

In the case at bar, Cytron is legally deficient to establish a prima facie case of anticipation against any of 1-8, 10, 14-25, 27 and 31-48. The basis for the anticipation rejections is set forth on pages 7-10 of the Final Action. As the Board may find, the Examiner's arguments with regard to Cytron are for the most part, very ambiguous and not-well reasoned, but based on out of context citation to certain teachings of Cytron that are seemingly off point or irrelevant to the claimed features.

Claims 1, 18 and 35 are clearly patentable and non-obvious over Cytron, in that Cytron does not disclose or suggest *analyzing program code to determine if there is at least one statement which can affect a desirability of performing at least one cache transaction, if the at*

least one statement is executed; determining a probability that the at least one statement will execute; determining the desirability of performing the at least one cache transaction based on the probability that the at least one statement will execute, as essentially claimed.

Cytron is directed to a compiler-directed cache coherence method for maintaining *cache coherence* in a multiprocessor system wherein each processor has a cache associated therewith and wherein the processors read contents of a shared memory location. In other words, Cytron is concerned with maintaining consistency of cached values in the various caches, which are read by an application. Although Cytron arguably discloses analyzing program code, there is nothing in Cytron that discloses the claimed invention of *analyzing the code to determine if there is a statement that can affect a desirability of performing a cache transaction, if the statement is executed*, and then *determining a probability that the statement will execute* and determining the desirability of performing a cache transaction *based on a probability that the statement will execute*.

With regard to ***analyzing program code to determine if there is at least one statement which can affect a desirability of performing at least one cache transaction, if the at least one statement is executed,*** the Examiner relies on page 231, 2.0 Algorithms, contending the "algorithm analyzes a program to determine the cachability of variables". However, it is not clear and the Examiner does not explain how, analyzing program code for "determining the cachability of variables" is even remotely related to ***analyzing program code to determine if there is at least one statement which can affect a desirability of performing at least one cache transaction, if the at least one statement is executed*** .

The Examiner further relies on page 230, section 1.1 as disclosing a "definition of cacheability or degree or desirability of caching". However, Cytron discloses a global address

space of a global memory (not cache memory) having addresses that are marked with a cachability status indicating the cachability of data at that address when a processor references the address. The Examiner does not explain how this even remotely relates to ***analyzing program code to determine if there is at least one statement which can affect a desirability of performing at least one cache transaction, if the at least one statement is executed***

The Examiner further relies on page 232, section 2.2 Posting Values to Global Memory, as disclosing "a program code sequence that should cause, i.e., highly desirable, a posting of a cached value, i.e., cache transaction". Once again, the Examiner does not explain how this even remotely relates to ***analyzing program code to determine if there is at least one statement which can affect a desirability of performing at least one cache transaction, if the at least one statement is executed*** Moreover, the Examiner's reliance in this regard is seemingly erroneous on its face as the Posting to Global memory algorithm relates to performing a global memory operation, not a cache operation.

Equally confusing is the basis for Examiner's contention that Cytron discloses the claimed process of ***determining a probability that the at least one statement will execute.*** The Examiner contends (on page 7 of the Final Action) that Cytron's execution model "analyzes DO loops to achieve parallelism, statements in a DO loop are always executed, i.e., probability of execution is determined to be 1." Again, the Examiner does not clearly explain how this even remotely relates to the claimed process of ***determining a probability that the at least one statement will execute.*** In fact, in the proper context of the claimed invention, the Examiner's argument seemingly makes no sense – based on the Examiner's analysis, the Examiner essentially asserts that Cytron teaches ***analyzing program code to determine if there is at least one statement*** (DO loop statement) ***which can affect a desirability of performing at least one***

cache transaction, if the at least one statement (DO loop statement) ***is executed,*** and ***determining a probability that the at least one statement*** (detected DO loop statement) ***will execute.*** Cytron's teachings simply do not support this.

The Examiner further contends that the claimed process of ***determining a probability that the at least one statement will execute*** is anticipated by Cytron's teachings in section 2.1, of processor-crossing dependencies, where the Examiner avers "the analyzer also determines a probability that a statement will execute in different processors …). This finding appears to be without merit and way off point as the Algorithm in section 2.1 is used to determine when the source and sink if a dependence execute in different processors – when two different statements execute on two different processors, one statement is a source of a given data variable and another statement is a sink of the given data variable. These cross dependences are used to determine where cache control points can be placed. Again, in the proper context of the claimed invention, the Examiner offers no reasoned argument as to how section 2.1., processor-crossing dependencies teaches ***analyzing program code to determine if there is at least one statement*** (which statement is the Examiner referring to??) ***which can affect a desirability of performing at least one cache transaction, if the at least one statement*** (which statement ??) ***is executed,*** and ***determining a probability that the at least one statement*** (which detected statement is the Examiner referring to??) ***will execute.***

As is evident from the above discussion, the Examiner's anticipation analysis with regard to Cytron is premised on out-of-context citations to different passages of Cytron coupled with unsupported conclusions as to how the elements of claims 1 and 18 are supported by such teachings. The Examiner's arguments are unsupported and leave Applicants only guessing as to

the basis for the Examiner's finding. There is simply no *prima facie* showing of anticipation of claims 1 and 18 based on Cytron.

G. The Combination of Cytron and Leving is Legally Deficient to Establish a *Prima Facie* Case of Obviousness Against the Claimed Inventions

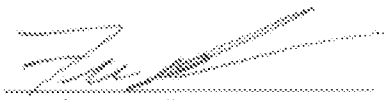
In rejecting claims under 35 U.S.C. 103, the Examiner bears the initial burden of presenting a prima facie case of obviousness. In re Rijckaert, 9 F.3d 1531, 1532 (Fed. Cir. 1993). The burden of presenting a prima facie case of obviousness is only satisfied by showing some objective teaching in the prior art or that knowledge generally available to one of ordinary skill in the art would lead that individual to combine the relevant teachings of the references. In re Fine, 837 F.2d 1071, 1074 (Fed. Cir. 1988). The test for obviousness is what the combined teachings of the applied prior art references would have suggested to one of ordinary skill in the art. In re Keller, 642 F.2d 413, 435; 208 U.S.P.Q. 871, 881 (CCPA 1981). The suggestion to combine the references should come from the prior art, and the Examiner cannot use hindsight gleaned from the invention itself to pick and choose among related prior art references to arrive at the claimed invention. In re Fine, 837 F.2d at 1075. If the Examiner fails to establish a prima facie case, the rejection is improper and must be overturned. In re Rijckaert, 9 F.3d at 1532 (citing In re Fine, 837 F.2d at 1074).

Claims 9 and 26 stand rejected as being unpatentable over Cytron in view of U.S. Patent No. 6,073,129 to Leving, et al. The rejection of claim 9 and 26 is based, in part, on the contention that Cytron discloses the elements of claims 1 and 18 from which claims 9 and 26 respectively depend. However, at the very least, the obviousness rejection is invalid by virtue of Cytron failing to disclose or suggest all elements of base claims 1 and 18. Further, it is

unquestionable that Levine fails to cure the deficiencies of Cytron with respect to claims 1 and 18 as discussed above. Accordingly, withdrawal of the obviousness rejections is requested.

II. CONCLUSION

Accordingly, for at least the above reasons, it is respectfully requested that the Board reverse all claim rejections under 35 U.S.C. §§ 101, 112, 102 and 103.



Frank V. DeRosa
Reg. No. 43,584

F. Chau & Associates, LLC
130 Woodbury Road
Woodbury, New York 11797
TEL: (516) 692-8888
FAX: (516) 692-8889

Claims Appendix

1. An automated method for managing a plurality of cachable entities, comprising the steps of:

analyzing program code to determine if there is at least one statement which can affect a desirability of performing at least one cache transaction, if the at least one statement is executed;

determining a probability that the at least one statement will execute;

determining the desirability of performing the at least one cache transaction based on the probability that the at least one statement will execute.

2. The method of claim 1, wherein the desirability of performing the at least one cache transaction is based on one of a frequency of access of at least one cachable entity, a size of at least one cachable entity, a time to one of fetch and materialize at least one cachable entity, a lifetime of at least one cachable entity, and a combination thereof.

3. The method of claim 1, wherein the at least one statement is a statement that modifies a value of at least one cachable entity, and wherein the desirability is based on an expected lifetime of the at least one cachable entity.

4. The method of claim 41, wherein the step of performing at least one cache transaction comprises one of storing at least one cachable entity in a cache, invalidating at least one cachable entity stored in a cache, updating at least one cachable entity stored in a cache, and a combination thereof.

5. The method of claim 1, further comprising the step of augmenting the program code with additional code to assist in determining the desirability of performing the at least one cache transaction.

6. The method of claim 41, further comprising the step of augmenting the program code with additional code to perform the at least one cache transaction.

7. The method of claim 4, wherein at least one of the step of invalidating the at least one cachable entity stored in the cache and the step of updating the at least one cachable entity stored in the cache comprise the step of performing data update propagation (DUP).

8. The method of claim 1, wherein the at least one statement is one of source code, assembly code, machine code, and structured query language (SQL) code.

9. The method of claim 8, wherein the at least one statement in the SQL code includes at least one SET statement.

10. The method of claim 1, wherein the cachable entities include query results.

11. The method of claim 1, wherein the analyzing step comprises the steps of:

detecting at least one query statement for retrieving at least one of the cachable entities from a cache;

generating a query key format; and

augmenting the program code with additional code for calculating a query key in accordance with the query key format.

12. The method of claim 11, further comprising performing at least one cache transaction, wherein performing comprises the steps of:

executing the augmented code to calculate the query key;

searching the cache using the query key; and

retrieving at least one cachable entity stored in the cache if the cachable entity corresponds to the query key.

13. The method of claim 12, further comprising the steps of:

processing the at least one query statement to retrieve at least one of the plurality of cachable entities, if there are no cachable entities in the cache which correspond to the query key;

storing the at least one retrieved cachable entity in the cache using the query key; and

associating at least one dependency with the at least one retrieved cachable entity.

14. The method of claim 1, wherein the at least one statement is a type that one of creates at least one cachable entity, deletes at least one cachable entity, and modifies a value of at least one cachable entity, wherein the analyzing step comprises the steps of:

generating an invalidation key format in accordance with the type of the at least one statement; and

augmenting the program code with additional code for calculating an invalidation key in accordance with the generated invalidation key format.

15. The method of claim 14, further comprising performing at least one cache transaction, wherein performing comprises the steps of:

executing the augmented code to calculate the invalidation key; and

invalidating at least one cachable entity stored in the cache that corresponds to the invalidation key.

16. The method of claim 15, wherein the step of invalidating at least one cachable entity comprises one of purging the cachable entity from the cache, purging the cachable entity from the cache and repopulating the cache, and updating the cache.

17. The method of claim 41, wherein the step of performing at least one cache transaction comprises the step of initializing a cache.

18. A program storage device readable by a machine, tangibly embodying a program of instructions executable by the machine to perform method steps for managing a plurality of cachable entities, the method steps comprising:

analyzing program code to determine if there is at least one statement which can affect a desirability of performing at least one cache transaction, if the at least one statement is executed;

determining a probability that the at least one statement will execute;

determining the desirability of performing the at least one cache transaction based on the probability that the at least one statement will execute.

19. The program storage device of claim 18, wherein the desirability of performing the at least one cache transaction is based on one of a frequency of access of at least one cachable entity, a size of at least one cachable entity, a time to one of fetch and materialize at least one cachable entity, a lifetime of at least one cachable entity, and a combination thereof.

20. The program storage device of claim 18, wherein the at least one statement is a statement that modifies a value of at least one cachable entity, and wherein the desirability is based on an expected lifetime of the at least one cachable entity.

21. The program storage device of claim 45, wherein the instructions for performing at least one cache transaction include instructions for one of storing at least one cachable entity in a cache, invalidating at least one cachable entity stored in a cache, updating at least one cachable entity stored in a cache, and a combination thereof.

22. The program storage device claim 18, further including instructions for augmenting the program code with additional code to assist in determining the desirability of performing the at least one cache transaction.

23. The program storage device of claim 45, further including instructions for augmenting the program code with additional code to perform the at least one cache transaction.

24. The program storage device of claim 21, wherein the instructions for at least one of invalidating the at least one cachable entity stored in the cache and updating the at least one cachable entity stored in the cache include instructions for performing data update propagation (DUP).

25. The program storage device of claim 18, wherein the at least one statement is one of source code, assembly code, machine code, and structured query language (SQL) code.

26. The program storage device of claim 25, wherein the at least one statement in the SQL code includes at least one SET statement.

27. The program storage device of claim 18, wherein the cachable entities include query results.

28. The program storage device of claim 18, wherein the instruction for performing the analyzing step include instructions for performing the steps of:

detecting at least one query statement for retrieving at least one of the cachable entities from a cache;

generating a query key format; and

augmenting the program code with additional code for calculating a query key in accordance with the query key format.

29. The program storage device of claim 28, further comprising instructions for performing at least one cache transaction, wherein the instructions for performing at least one cache transaction include instructions for performing the steps of:

executing the augmented code to calculate the query key;

searching the cache using the query key; and

retrieving at least one cachable entity stored in the cache if the cachable entity corresponds to the query key.

30. The program storage device of claim 29, further including instructions for performing the steps of:

processing the at least one query statement to retrieve at least one of the plurality of cachable entities, if there are no cachable entities in the cache which correspond to the query key;

storing the at least one retrieved cachable entity in the cache using the query key; and

associating at least one dependency with the at least one retrieved cachable entity.

31. The program storage device of claim 18, wherein the at least one statement is a type that one of creates at least one cachable entity, deletes at least one cachable entity, and modifies a value of at least one cachable entity, wherein the instructions for performing the analyzing step include instructions for performing the steps of:

generating an invalidation key format in accordance with the type of the at least one statement; and

augmenting the program code with additional code for calculating an invalidation key in accordance with the generated invalidation key format.

32. The program storage device of claim 31, further comprising instructions for performing at least one cache transaction, wherein the instructions for performing at least one cache transaction include instructions for performing the steps of:

executing the augmented code to calculate the invalidation key; and

invalidating at least one cachable entity stored in the cache that corresponds to the invalidation key.

33. The program storage device of claim 32, wherein the instructions for invalidating at least one cachable entity include instructions for performing one of purging the cachable entity from the cache, purging the cachable entity from the cache and repopulating the cache, and updating the cache.

34. The program storage device of claim 45, wherein the instructions for performing the at least one cache transaction include instructions for initializing a cache.

35. A system for managing a plurality of cachable entities, comprising:

a program analyzer to analyze program code and determine if there is at least one statement which can affect a desirability of performing at least one cache transaction, if the at least one statement is executed;

the program analyzer determining a probability that the at least one statement will execute and determining the desirability of performing the at least one cache transaction based on the probability that the at least one statement will execute; and

a cache manager for performing the at least one cache transaction if it is determined to be desirable.

36. The system of claim 35, wherein the desirability of performing the at least one cache transaction is based on one of a frequency of access of at least one cachable entity, a size of at least one cachable entity, a time to one of fetch and materialize at least one cachable entity, a lifetime of at least one cachable entity, and a combination thereof.

37. The system of claim 35, wherein the at least one detected statement is a statement that modifies a value of at least one cachable entity, and wherein the desirability is based on an expected lifetime of the at least one cachable entity.

38. The system of claim 35, wherein the cache manager performs one of storing at least one cachable entity in the cache, invalidating at least one cachable entity stored in the cache, updating at least one cachable entity stored in the and a combination thereof.

39. The system of claim 35, wherein the cache manager augments the program code with additional code to assist in determining the desirability of performing the at least one cache transaction.

40. The system of claim 35, wherein the cache manager augments the program code with additional code to perform the at least one cache transaction.

41. The method of claim 1, further comprising performing the at least one cache transaction if it is determined to be desirable.

42. The method of claim 1, further comprising determining said probability based on a likelihood of a value of a cachable entity changing.

43. The method of claim 42, further comprising caching a value of said cachable entity, if said probability is less than or equal to a threshold.

44. The method of claim 42, further comprising invalidating or updating a value of said cachable entity, if said probability is greater than or equal to a threshold.

45. The program storage device of claim 18, further comprising instructions for performing the at least one cache transaction if it is determined to be desirable.

46. The program storage device of claim 18, further comprising instructions for determining said probability based on a likelihood of a value of a cachable entity changing.

47. The program storage device of claim 46, further comprising instructions for caching a value of said cachable entity, if said probability is less than or equal to a threshold.

48. The program storage device of claim 46, further comprising instructions for invalidating or updating a value of said cachable entity, if said probability is greater than or equal to a threshold.

49. The method of claim 1 further comprising the step of performing the at least one cache transaction if said desirability is greater than or equal to a threshold.

50. The program storage device of claim 18, further comprising instructions for performing the at least one cache transaction if said desirability is greater than or equal to a threshold.

51. The system of claim 35, wherein the cache manager performs the at least one cache transaction if said desirability is greater than or equal to a threshold.

Evidence Appendix

There is no evidence submitted pursuant to 37 CFR §§ 1.130, 1.131 or 1.132 or any other evidence entered by the examiner and relied upon by appellant in this Appeal.

Related Proceedings Appendix

None.